Stanford University Computer Science Department
2007 Comprehensive Exam in Databases

- The exam is *open book and notes*, but not open computer.
- There are six problems on the exam, with a varying number of points for each problem and subproblem for a total of 60 points (i.e., one point per minute). It is suggested that you look through the entire exam before getting started, in order to plan your strategy.
- Please write your solutions in the spaces provided on the exam. Make sure your solutions are neat and clearly marked.
- *Simplicity and clarity of solutions will count.* You may get as few as 0 points for a problem if your solution is far more complicated than necessary, or if we cannot understand your solution.

MAGIC NUMBER: _____

Problem	1	2	3	4	5	6	TOTAL
Max. points	10	8	12	18	3	9	60
Points							

1. *Data Modeling* **(10 points)**

   Design an Entity/Relationship diagram for a database that holds the following information:

   i. There are teams. Each team has a unique team name, a city in which it plays, and a mascot.

   ii. There are players. Each player has a name, a number, and a position. Each player plays for a unique team.

   iii. Teams assign unique numbers to their players, but two teams may use the same numbers.

   iv. Each team has at most one team captain, who is a player.

   v. The day on which two teams play, and which team is the home team if they do play. You may assume no pair of teams plays more than once.

   Your diagram should be as succinct as possible, but must capture all the above information. If you find the specifications ambiguous, appeal to the way the world actually works, e.g., it is entirely possible that two players have the same name.

2. *Dependency Theory* **(8 points)**

   The following functional dependencies hold in relation $R(A, B, C, D, E, F)$:

   $AB \rightarrow C$
   $DE \rightarrow F$
   $F \rightarrow B$
   $C \rightarrow E$

   Find all the keys for R. Explain your reasoning for partial credit.

3. *Relational Algebra* **(12 points)**

   Suppose we have a relation $T(P, C, I)$ representing a tree. That is, nodes are represented by integers, and the meaning of a tuple (p, c, i) in T is that node c is the ith child of node p, from the left. We want you to write certain queries in **relational algebra**, using only the basic operations: select, project, natural and theta join, product, renaming, union, intersection, and difference. You may define temporary relations if you like; i.e., you may break an expression into a sequence of steps. To receive full credit, your expressions should be (close to) as simple as possible.

   (a) (3 pts.) Write a query to produce the set of distinct pairs of nodes that are siblings (i.e., they have the same parent).

(b) (3 pts.) If you can use arithmetic in selection conditions, e.g., $x = y + 1$, then it is relatively easy to write a query that produces the set of pairs (a, b) such that b is the right-sibling of a; that is, a and b have the same parent, and b is immediately to the right of a among the children of their parent. Write this query.

(c) (6 pts.) But you do not need arithmetic to write the query of part (b). Write the same query, using only conditions of the form $x\theta y$, where θ is a single comparison operator (e.g., = or $<$) and x and y are attributes or constants. Both selection and theta-join are limited in this way.

4. *SQL* **(18 points, 3 each part)**

Two queries are *equivalent on a database instance* if they produce the same result relation when evaluated over that instance.

Consider a SQL database with two tables, $R(A, B)$ and $S(C)$. Assume NULL values are not permitted in any attribute; make no other assumptions about the data.

Below are pairs of queries. For each pair, select one of the following three options:

- **EQ** – The two queries are equivalent on every possible instance of R and S.
- **NEQ** – There are no instances of R and S on which the two queries are equivalent.
- **SEQ** – The two queries are equivalent for some instances of R and S, but not all. When choosing this option, state conditions over the data in R and S that ensure the two queries are equivalent. Try to make the conditions as general as possible, meaning whenever the conditions are violated the queries may not be equivalent. (E.g., "The tables are empty" is not a good choice.) The conditions should be stated succinctly and should use SQL constraint terminology (e.g., "A is a key") when applicable.

(a)
```
Query1: Select A From R
Query2: Select A From R, S
```

EQ, **NEQ**, or **SEQ** with condition:

(b)
```
Query1: Select A From R
Query2: Select A From R, S Where B = C
```

EQ, **NEQ**, or **SEQ** with condition:

(continued on next page)

(c)
```
Query1: Select Distinct A From R
Query2: Select A From R Group By A
```

EQ, **NEQ**, or **SEQ** with condition:

(d)
```
Query1: Select count(*) From R
Query2: Select count(Distinct A) From R
```

EQ, **NEQ**, or **SEQ** with condition:

(e)
```
Query1: Select max(A) From R
Query2: Select A From R Where A >= all (select A from R)
```

EQ, **NEQ**, or **SEQ** with condition:

(f)
```
Query1: Select Distinct A From R Group By A Having max(B) > 10
Query2: Select Distinct A from R Where B > 10
```

EQ, **NEQ**, or **SEQ** with condition:

5. *Views* **(3 points)**

Is it possible to create an index on a SQL view? Answer *yes* or *no*, with a brief explanation.

6. *Transactions* **(9 points, 3 each part)**

Consider two tables $R(A, B)$ and $S(C)$ as in Problem 4. Below are pairs of transactions. For each pair, decide whether it is possible for *nonserializable* behavior to be exhibited when executing the transactions together, while respecting their specified isolation levels. Assume each transaction completes successfully.

(a)
```
Transaction 1: {
  Set Transaction Isolation Level Read Committed;
  Select count(*) From R;
  Select count(*) From S;
  Commit;
  }
Transaction 2: {
  Set Transaction Isolation Level Serializable;
  Insert Into R Values (1,2);
  Insert Into S Values (3);
  Commit;
  }
```

Nonserializable behavior possible (*yes* or *no*)? [ ]

(b)
```
Transaction 1: {
  Set Transaction Isolation Level Read Committed;
  Select count(*) From R;
  Select count(*) From S;
  Commit;
  }
Transaction 2: {
  Set Transaction Isolation Level Serializable;
  Insert Into R Values (1,2);
  Insert Into R Values (3,4);
  Commit;
  }
```

Nonserializable behavior possible (*yes* or *no*)? [ ] ** *one more on next page*

(c)
```
Transaction 1: {
  Set Transaction Isolation Level Repeatable Read;
  Select count(*) From R;
  Select count(*) From S;
  Select count(*) From R;
  Commit;
}
Transaction 2: {
  Set Transaction Isolation Level Serializable;
  Insert Into R Values (1,2);
  Commit;
  }
```

Nonserializable behavior possible (*yes* or *no*)? [ ]